Drawings

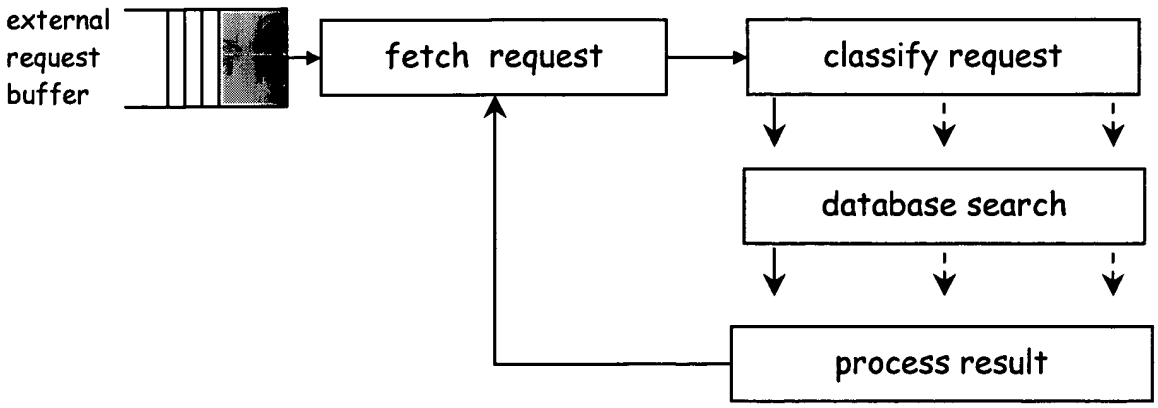

Figure 1: Transaction Processing System (Prior Art).

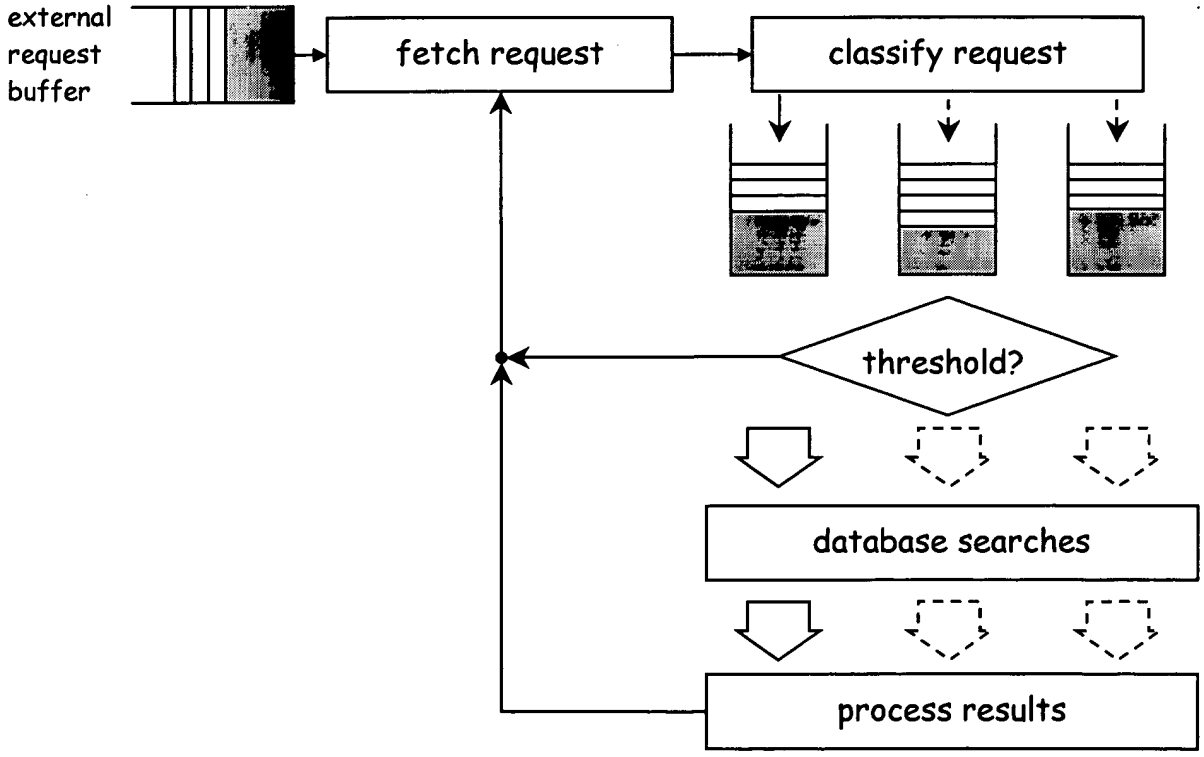

Figure 2: Transaction Processing System with Request Buffering.

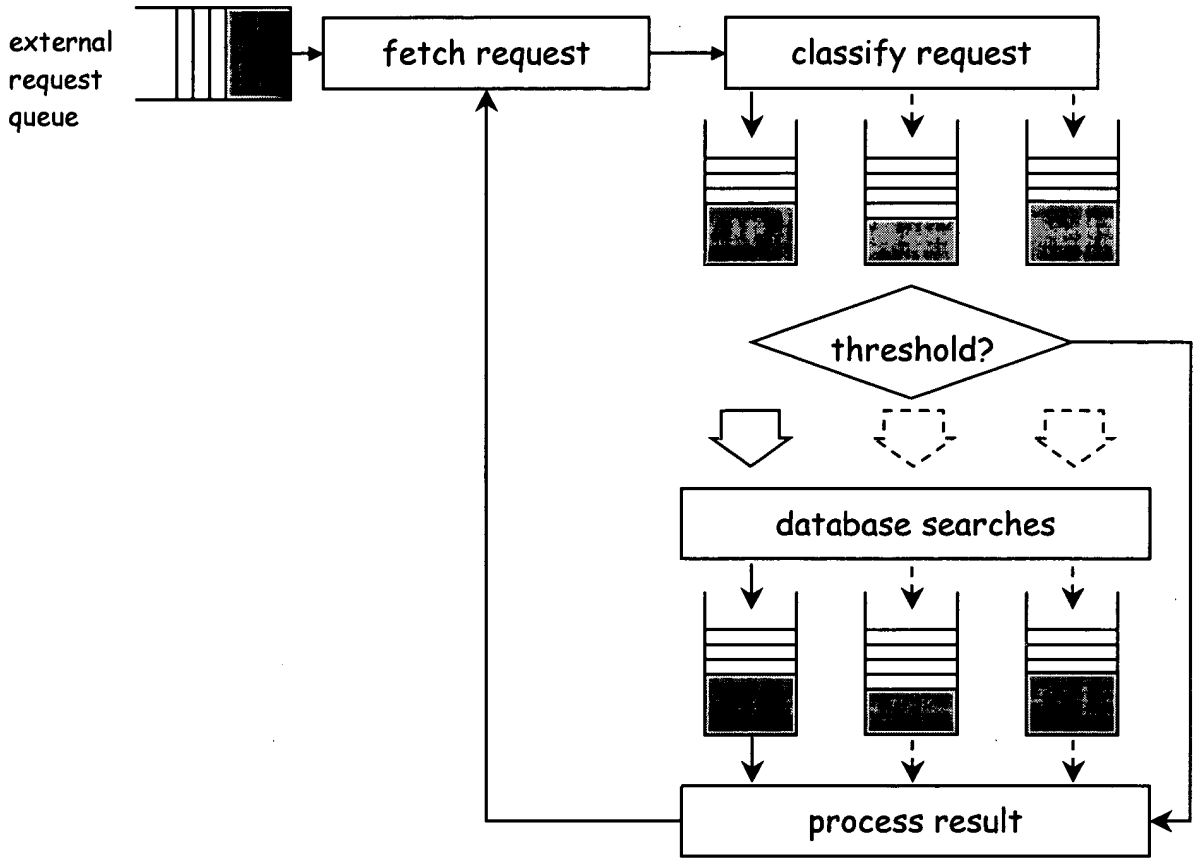

Figure 3: Transaction Processing System with Request and Result Buffering.

First Set of Search Requests

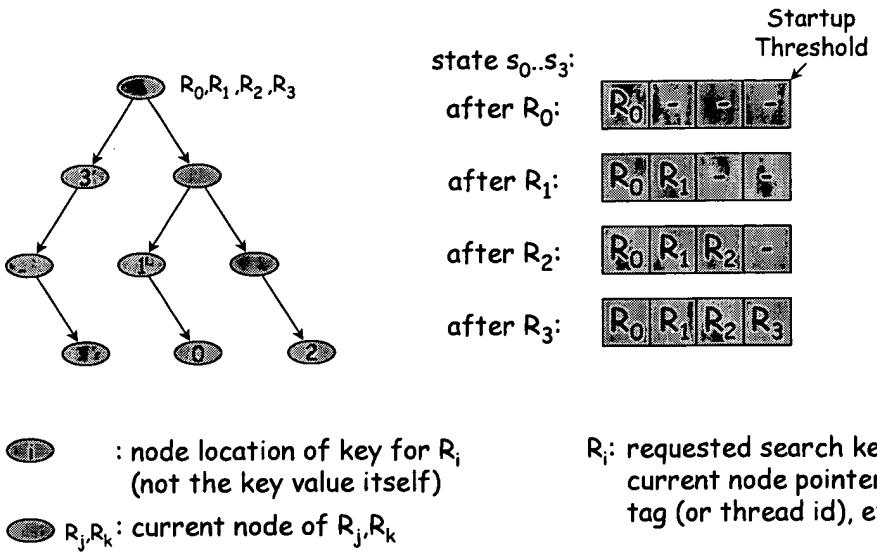

Figure 4: Example of a tree traversal buffering.

First Pipelined Search

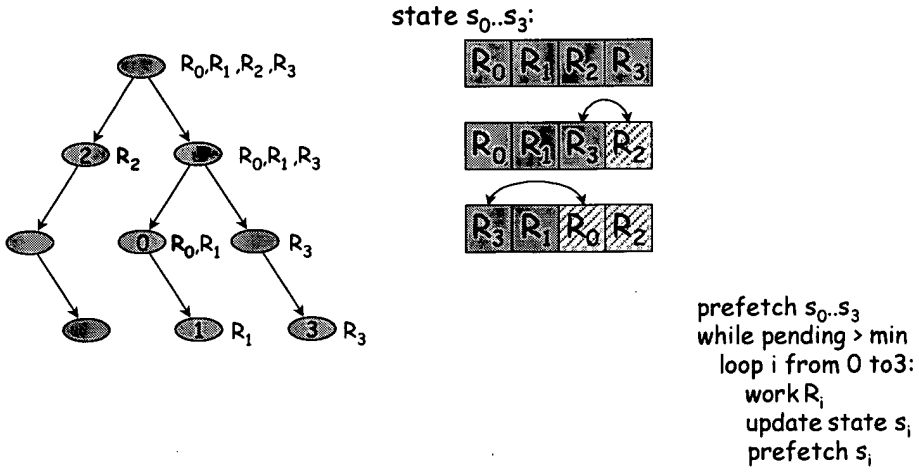

Figure 5: Example of a pipelined tree search traversal.

Second Pipelined Search

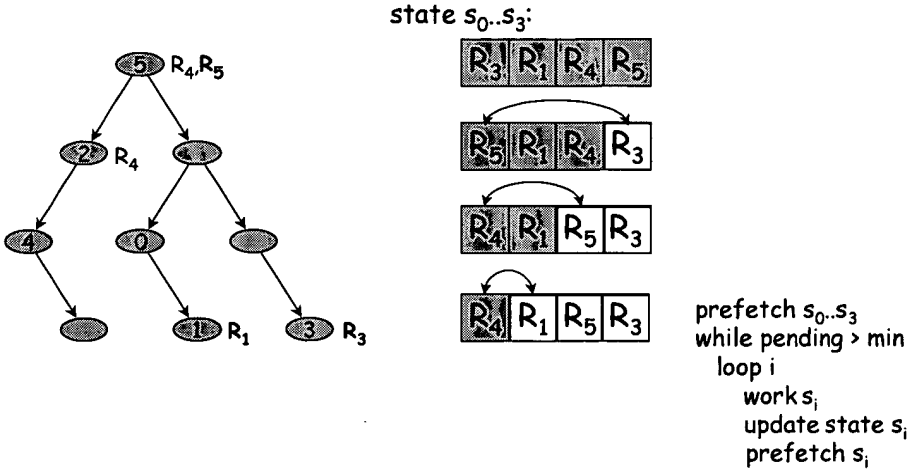

Figure 6: Example of a pipelined tree search traversal state.

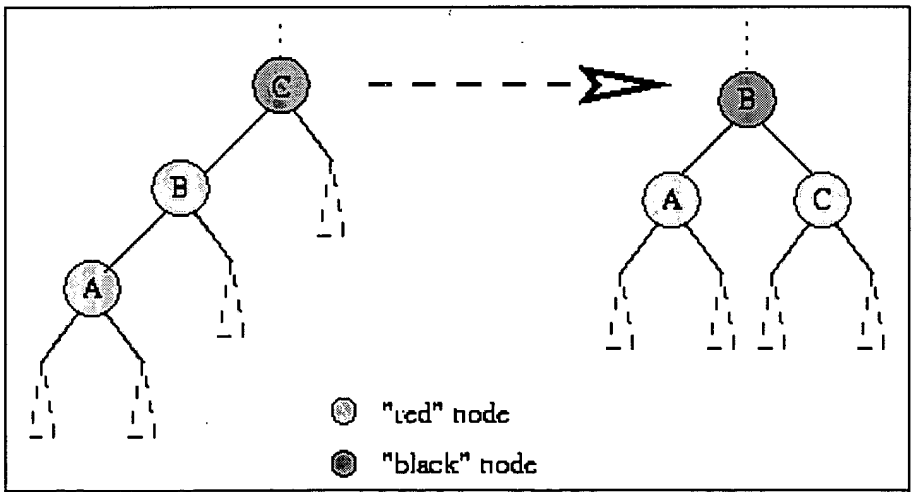

Figure 7: Red-Black Tree Insertion.

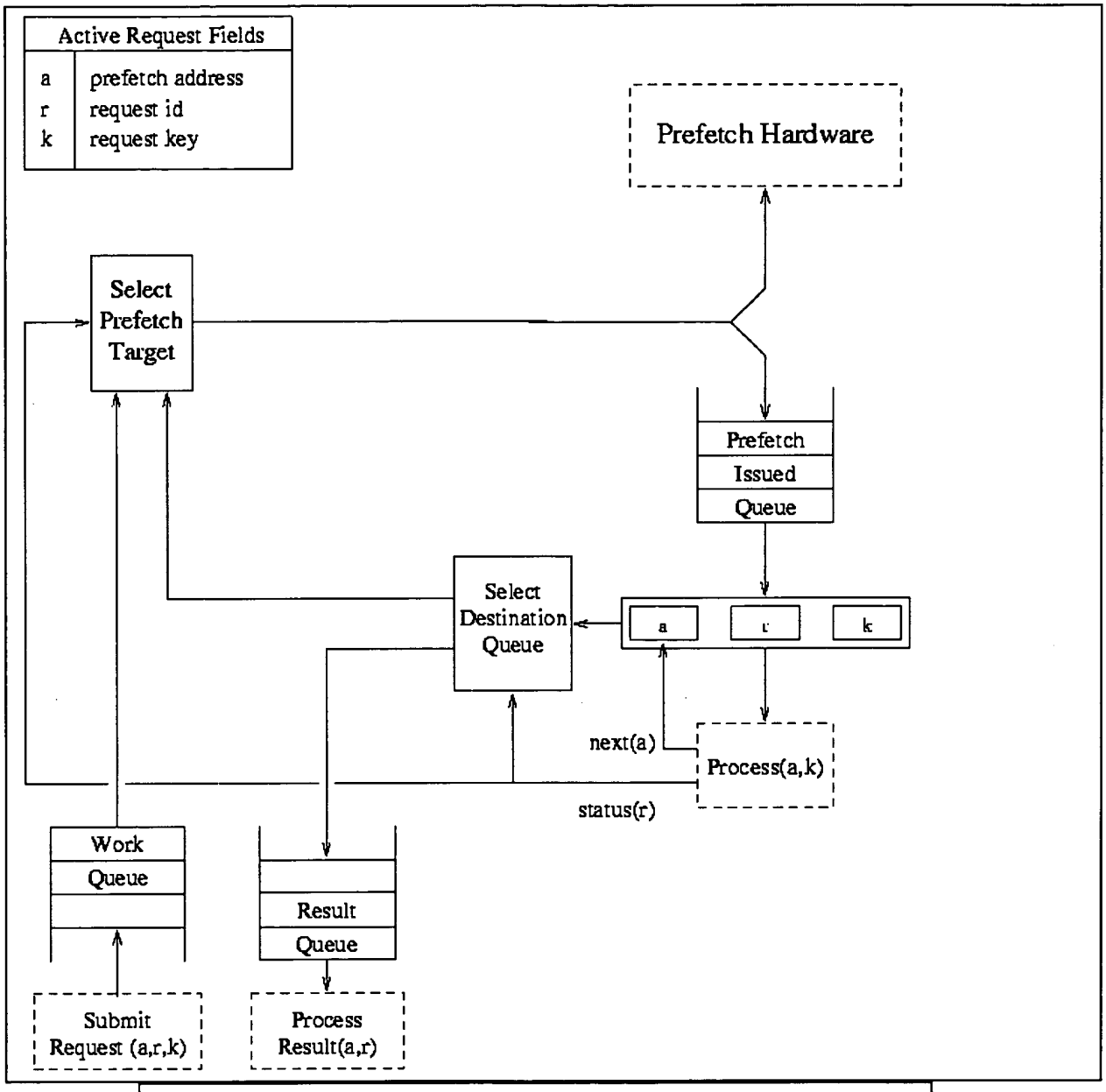

Figure 8: Restructuring mechanism, as implemented in software.

```
RESTRUCTURED-TRAVERSAL( S, request )
begin
        AQ.enqueue( request );
        if AQ.size ≥ K then
                SOFTWARE-PIPELINE( S, AQ, RQ );
        if RQ.size = 0 then
                return POSTPONE
        else
                return RQ.dequeue()
end
```

Figure 9: Accumulating K requests on accumulation queue AQ for software pipelined traversals of data structure S, where K is the startup threshold. Accumulated results are turned from result queue RQ.

```
TREE-DELAYED-SEARCH( lower )
begin
        integer i, prologue;

        prologue ←MIN(lower, RQ.size);
        i ← 0;
        while i<prologue do
                PREFETCH( RQ.elem[i] );
                i ← i + 1;
        end while
        TREE-RECURSIVE-SEARCH( lower );
end
```

Figure 10: Recursive search requests, initial pre-recursive component.

```
TREE-RECURSIVE-SEARCH( lower )
begin
        i ← 0;
        while i<AQ.size do
                request ← AQ.elem[i];
                k ← request.key;
                n ← request.node;
                if n = NIL or k = n.key then
                        AQ.delete( request );
                        RQ.enqueue( request );
                else
                        if k < n.key then  request.node ← n.left;
                                else  request.node ← n.right;
                        endif
                        PREFETCH( request.node );
                endif
                i ← i + 1;
        end while
        if AQ.size ≥ lower then TREE-RECURSIVE-SEARCH( lower ); endif
end
```

Figure 11: Recursive search requests, recursive component.

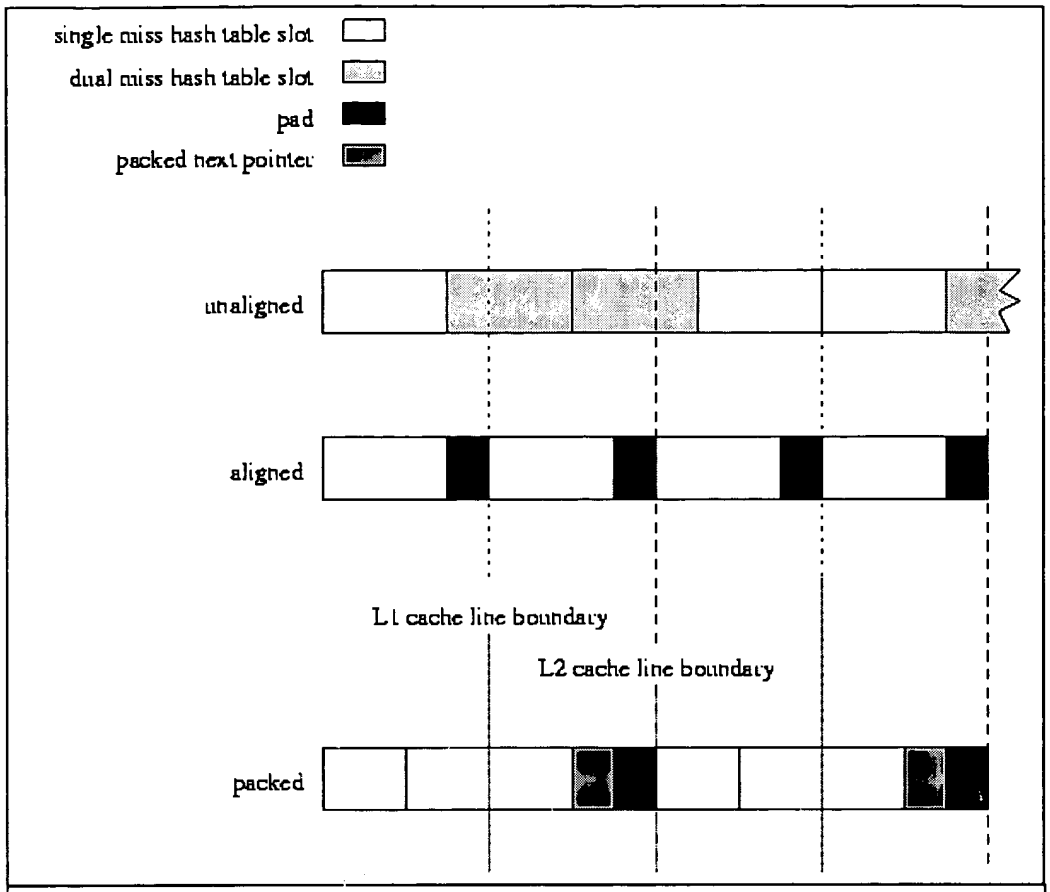

Figure 12: Alignment of Hash Table entries on cache line boundaries.

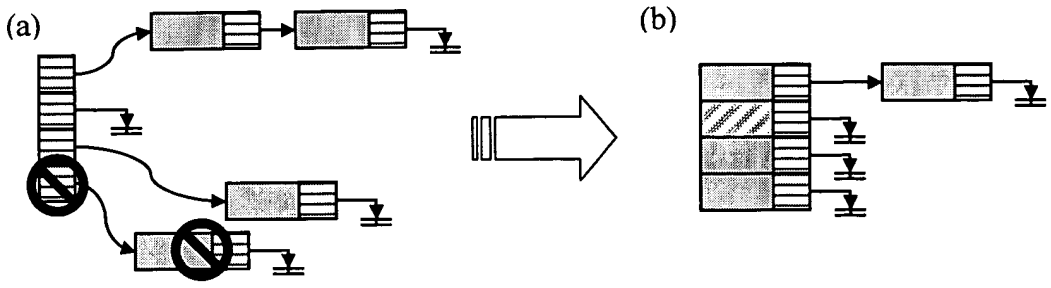

Figure 13: Hash Table homogenezation.

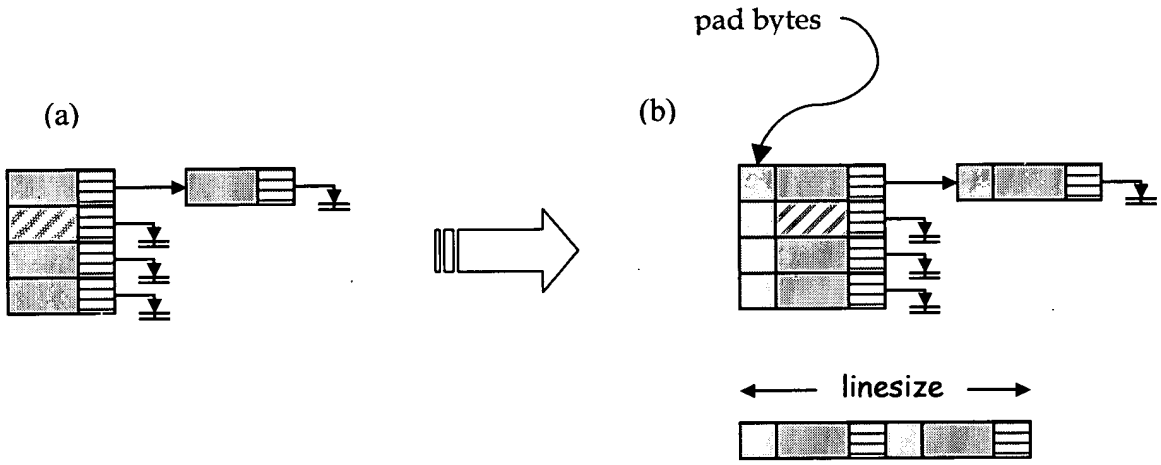

Figure 14: Hash Table padding.

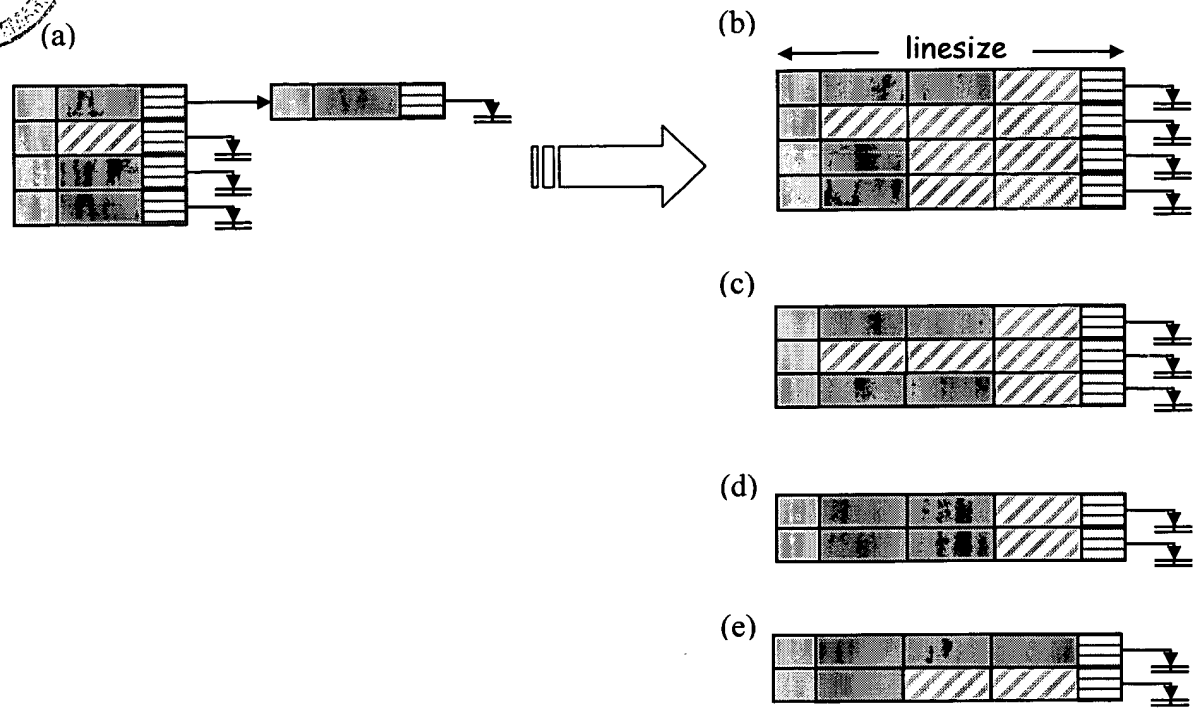

Figure 15: Hash table packing. Representing a homogeneous hash table structure (a) as a packed structure (b), which can be re-balanced to make the table less sparse as in (c), (d), or (e).

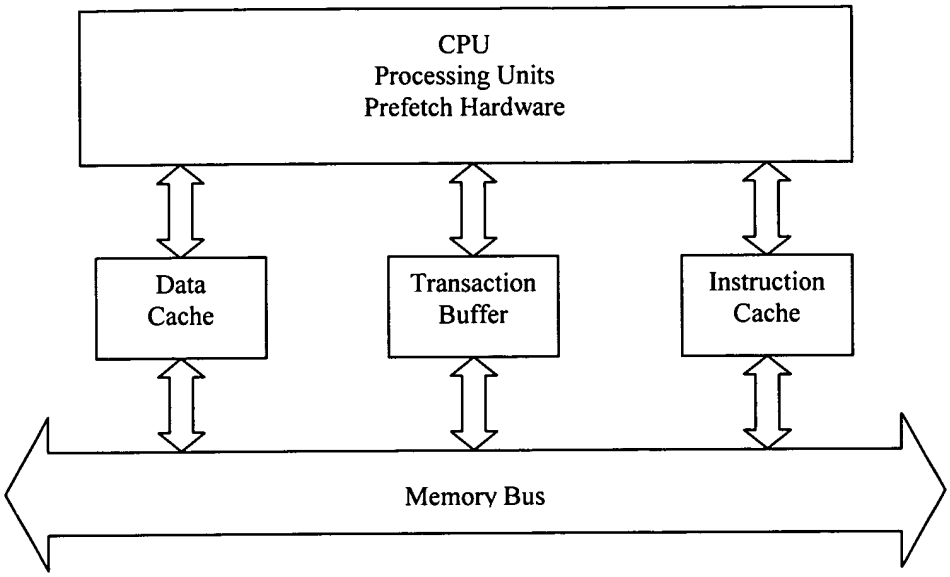

Figure 16: Transaction Buffer.

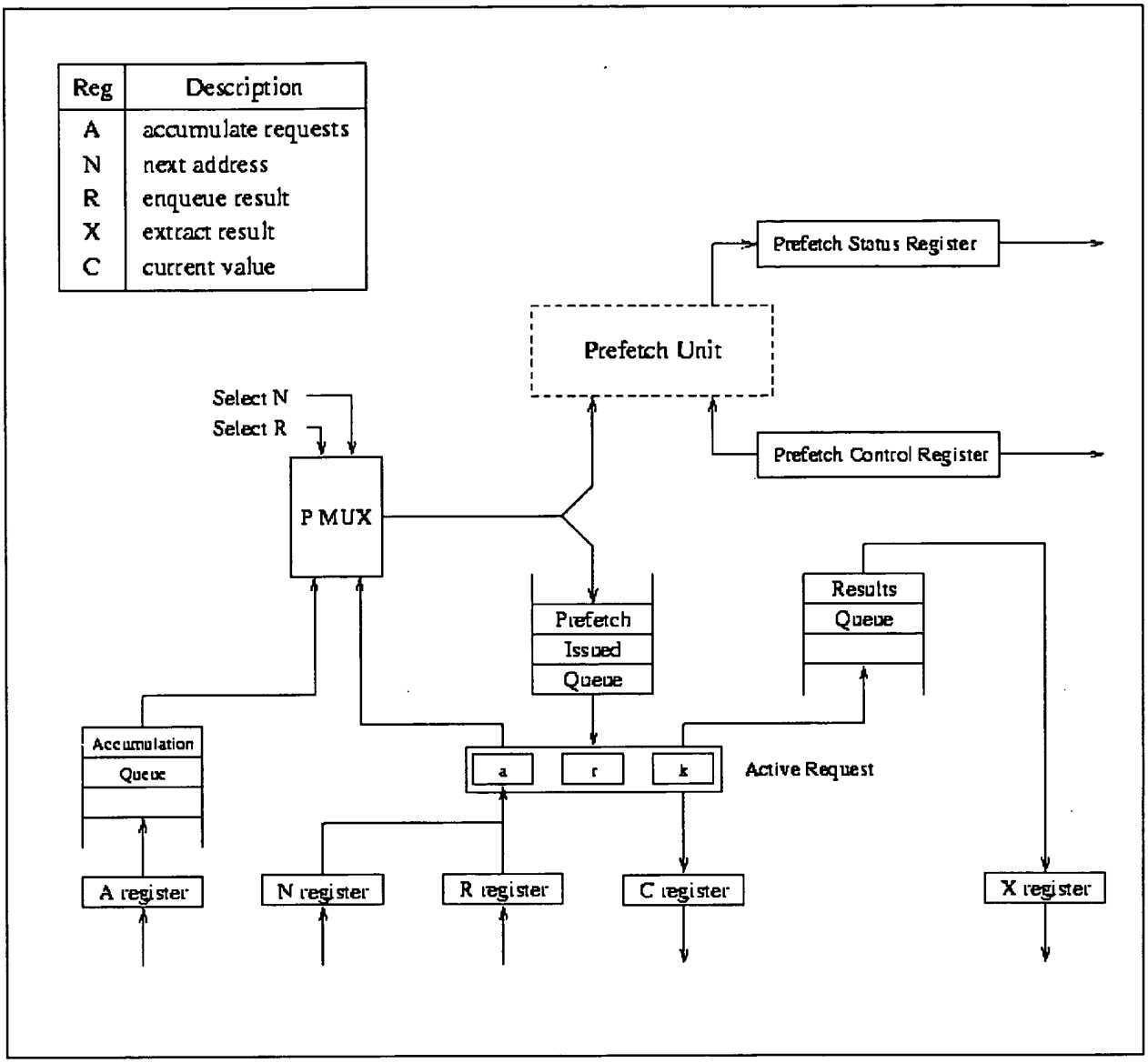

Figure 17: Transaction Buffer Details, single set of queues.